Fast TRAC: A Parameter-Free Optimizer for Lifelong Reinforcement Learning

Aneesh Muppidi
Harvard College
aneeshmuppidi@college.harvard.edu

Zhiyu Zhang
Harvard University
zhiyuz@seas.harvard.edu

Heng Yang
Harvard University
hankyang@seas.harvard.edu

Abstract

A key challenge in lifelong reinforcement learning (RL) is the loss of plasticity, where previous learning progress hinders an agent's adaptation to new tasks. While regularization and resetting can help, they require precise hyperparameter selection at the outset and environment-dependent adjustments. Building on the principled theory of online convex optimization, we present a parameter-free optimizer for lifelong RL, called TRAC, which requires no tuning or prior knowledge about the distribution shifts. Experiments on Procgen and Gym Control environments show that TRAC works surprisingly well—mitigating loss of plasticity and rapidly adapting to challenging distribution shifts—despite the underlying optimization problem being nonconvex and nonstationary. The code to install TRAC and run experiments can be found here.

1 Introduction

Spot, the agile robot dog, has been learning to walk confidently across soft, lush grass. But when Spot moves to a rocky gravel surface, she stumbles. When Spot tries to walk across a sandy beach or on ice, the challenges multiply. Spot wants to adapt quickly to these new terrains, but she never knows when the terrain will change and how different it will be.

Spot's struggle exemplifies a real-world decision making challenge, known as *lifelong reinforcement learning* (lifelong RL), where an agent must continually learn to handle the nonstationarity of the environment. At first glance, there appears to be an obvious solution: given a policy gradient oracle, the agent could just keep running gradient descent nonstop. However, recent experiments have demonstrated an intriguing behavior called *loss of plasticity* (Dohare et al., 2021; Lyle et al., 2022; Abbas et al., 2023; Sokar et al., 2023; Nikishin et al., 2022; Ahn et al., 2024): despite persistent gradient steps, an agent can gradually lose its responsiveness to incoming observations.

From the optimization perspective, loss of plasticity might be attributed to the *lack of stability* under gradient descent. That is, the weights of the agent's parameterized policy can drift far away from the origin (or a good initialization), leading to a variety of undesirable behaviors.[1] Adding an L_2 regularizer to the optimization objective (Kumar et al., 2023) or periodically resetting the weights (Dohare et al., 2021; Asadi et al., 2023; Sokar et al., 2023; Ahn et al., 2024) can mitigate the problem. However, these methods rely on hyperparameters like regularizer magnitude and resetting frequency, which must be tuned for each environment—a challenging task incompatible with lifelong RL's one-shot nature. This challenge motivates our contributions.

[1]Such as the inactivation of many neurons, due to dead ReLU activations (Abbas et al., 2023; Sokar et al., 2023).

Contribution

- **Algorithm:** Building on a series of results in OCO (Cutkosky & Orabona, 2018; Cutkosky, 2019; Cutkosky et al., 2023; Zhang et al., 2024), we propose a (hyper)-*parameter-free* optimizer for lifelong RL, called **TRAC** (Adap**T**ive **R**egulariz**A**tion in **C**ontinual environments). Intuitively, the idea is a refinement of regularization: instead of manually selecting the magnitude of regularization beforehand, TRAC chooses that in an online manner without any hyperparameter tuning.

- **Experiment** Using an instantiation of TRAC with *Proximal Policy Optimization* (PPO) called TRAC PPO, we evaluate on lifelong settings of Procgen and Gym Control. In settings where existing approaches (Abbas et al., 2023; Kumar et al., 2023) struggle, we find that TRAC PPO mitigates loss of plasticity and rapidly adapts when new distribution shifts are introduced. Such findings might be surprising: the theoretical advantage of TRAC is motivated by the convexity in OCO, but lifelong RL is *both nonconvex and nonstationary* in terms of optimization.

2 Lifelong RL

As a sequential decision making framework, *reinforcement learning* is commonly framed as a *Markov Decision Process* (MDP) defined by the state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition dynamics $P(s_{t+1}|s_t, a_t)$, and the reward function $R(s_t, a_t, s_{t+1})$. In the t-th round, starting from a state $s_t \in \mathcal{S}$, the learning agent needs to choose an action $a_t \in \mathcal{A}$ without knowing P and R. Then, the environment samples a new state $s_{t+1} \sim P(\cdot|s_t, a_t)$, and the agent receives a *reward* $r_t = R(s_t, a_t, s_{t+1})$. From a practical perspective, we measure the agent's performance by its cumulative reward $\sum_{t=1}^{T} r_t$. This standard setting concerns a *stationary* MDP. The present work studies a nonstationary variant called *lifelong* RL, where the transition dynamics P_t and the reward function R_t can vary over time. We implicitly assume P_t and R_t to be *piecewise constant* over time, and each piece is called a *task*.

Lifelong RL as online optimization Modern RL methods, like PPO (Schulman et al., 2017), utilize *policy parameterization* with a weight vector $\theta_t \in \mathbb{R}^d$. After sampling a_t and receiving new observations, the agent defines a *loss function* $J_t(\theta)$ to evaluate each weight θ. The *policy gradient* $g_t = \nabla J_t(\theta_t)$ is then computed, and a *first order optimization algorithm* OPT updates the weight: $\theta_{t+1} = \mathrm{OPT}(\theta_t, g_t)$. We treat the environment's feedback as a *policy gradient oracle* $\mathcal{G}$, mapping time t and weight θ_t to a policy gradient $g_t = \mathcal{G}(t, \theta_t)$. Our goal is to design an optimizer OPT suited for lifelong RL.

3 Method

Inspired by (Cutkosky et al., 2023), we study lifelong RL by exploiting its connection to *Online Convex Optimization* (OCO; Zinkevich, 2003). OCO is a key problem in online learning, with significant efforts to design *parameter-free* algorithms requiring minimal tuning (Orabona & Pál, 2016; Foster et al., 2017; Cutkosky & Orabona, 2018; Jacobsen & Cutkosky, 2022). The surprising observation of Cutkosky et al. (2023) is that several algorithmic ideas closely tied to the convexity of OCO can actually improve nonconvex deep learning training, suggesting certain notions of "near convexity" on its loss landscape. We find that lifelong RL (which is *both nonconvex and nonstationary* in terms of optimization) exhibits a similar behavior.

Basics of (parameter-free) OCO OCO concerns a sequential optimization problem where the convex loss function l_t can vary arbitrarily over time. In the t-th iteration, the optimization algorithm picks an iterate x_t and then observes a gradient $g_t = \nabla l_t(x_t)$. Motivated by the pursuit of "convergence" in optimization, the standard objective is to guarantee low (i.e., sublinear in T) *static regret*, defined as

$$\mathrm{Regret}_T(l_{1:T}, u) := \sum_{t=1}^{T} l_t(x_t) - \sum_{t=1}^{T} l_t(u),$$

where T is the total number of rounds, and u is a *comparator* that the algorithm does not know beforehand. In other words, the goal is to make $\mathrm{Regret}_T(l_{1:T}, u)$ small for *all* possible loss sequence $l_{1:T}$ and comparator u.

Parameter-free algorithms bound $\mathrm{Regret}_T(l_{1:T}, u)$ directly (without taking the maximum) by a function of both $l_{1:T}$ and u. With this refined bound, there is no need to pick an uncertainty set $\mathcal{U}$, and much less hyperparameter tuning is needed (Orabona, 2023, Chapter 9).

TRAC for lifelong RL Now back to lifelong RL. As we discussed, a fundamental challenge here is the excessive drifting of the weights θ_t, and this can be fixed by enforcing the proximity to a good reference point θ_{ref}. Different from existing approaches like L_2 regularization and resetting, parameter-free OCO provides a principled solution to this problem without hyperparameter-tuning. Naming this algorithm as TRAC, we present its generic template as Algorithm 1, which calls Algorithm 2 (a one-dimensional scale tuner) as the key subroutine.

From the technical perspective, TRAC assembles three techniques in parameter-free OCO: the *direction-magnitude decomposition* from (Cutkosky & Orabona, 2018), the *additive aggregation* from (Cutkosky, 2019), and the erfi *potential function* from (Zhang et al., 2024).

Algorithm 1 TRAC: Parameter-free Adaption for Continual Environments.

```
1: Input: A policy gradient oracle $\mathcal{G}$; a first order optimization algorithm BASE; a reference point
   $\theta_{\mathrm{ref}} \in \mathbb{R}^d$; $n$ discount factors $\beta_1, \dots, \beta_n \in (0, 1]$ (default: $0.9, 0.99, \dots, 0.999999$).
2: Initialize: Create $n$ copies of Algorithm 2, denoted as $\mathcal{A}_1, \dots, \mathcal{A}_n$. For each $j \in [1:n]$, $\mathcal{A}_j$ uses
   the discount factor $\beta_j$. Initialize the algorithm BASE at $\theta_{\mathrm{ref}}$. Let $\theta_1 = \theta_{\mathrm{ref}}$.
3: for $t = 1, 2, \dots$ do
4:     Obtain the $t$-th policy gradient $g_t = \mathcal{G}(t, \theta_t) \in \mathbb{R}^d$.
5:     Send $g_t$ to BASE as its $t$-th input, and get its output $\theta_{t+1}^{\mathrm{Base}} \in \mathbb{R}^d$.
6:     For all $j \in [1:n]$, send $\langle g_t, \theta_t - \theta_{\mathrm{ref}} \rangle$ to $\mathcal{A}_j$ as its $t$-th input, and get its output $s_{t+1,j} \in \mathbb{R}$.
7:     Define the scaling parameter $S_{t+1} = \sum_{j=1}^{n} s_{t+1,j}$.
8:     Update the weight of the policy,
           $$\theta_{t+1} = \theta_{\mathrm{ref}} + \left(\theta_{t+1}^{\mathrm{Base}} - \theta_{\mathrm{ref}}\right) S_{t+1}.$$
9: end for
```

Algorithm 2 1D Discounted Tuner of TRAC.

```
1: Input: Discount factor $\beta \in (0, 1]$; small value $\varepsilon > 0$ (default: $10^{-8}$).
2: Initialize: The running variance $v_0 = 0$; the running (negative) sum $\sigma_0 = 0$.
3: for $t = 1, 2, \dots$ do
4:     Obtain the $t$-th input $h_t$.
5:     Let $v_t = \beta^2 v_{t-1} + h_t^2$, and $\sigma_t = \beta \sigma_{t-1} - h_t$.
6:     Select the $t$-th output
           $$s_{t+1} = \frac{\varepsilon}{\mathrm{erfi}(1/\sqrt{2})} \mathrm{erfi}\left(\frac{\sigma_t}{\sqrt{2 v_t} + \varepsilon}\right),$$
       where erfi is the imaginary error function queried from standard software packages.
7: end for
```

Without going deep into the theory, here is an overview of the important ideas.

First, TRAC is a meta-algorithm that operates on top of a "default" optimizer BASE. It can simply be gradient descent with a constant learning rate, or ADAM (Kingma & Ba, 2014) as in our experiments. Applying BASE alone would be equivalent to enforcing the scaling parameter $S_{t+1} \equiv 1$ in TRAC, but this would suffer from the drifting of $\theta_{t+1}^{\mathrm{Base}}$ (and thus, the weight θ_{t+1}). To fix this issue, TRAC uses the tuner (Algorithm 2) to select the scaling parameter S_{t+1}, making it *data-dependent*. Typically S_{t+1} is within $[0, 1]$ (see Figure 10 to 12), therefore essentially, we define the updated weight θ_{t+1} as a *convex combination* of the BASE's weight θ_t^{Base} and the reference point θ_{ref},

$$\theta_{t+1} = S_{t+1} \cdot \theta_{t+1}^{\text{Base}} + (1 - S_{t+1})\theta_{\text{ref}}.$$

This brings the weight closer to θ_{ref}, which is known to be "safe" (i.e., not overfitting any particular lifelong RL task), although possibly conservative. To inject appropriate conservatism without hyperparameter tuning, the tuner (Algorithm 2) uses the erfi function decision rule (theoretically optimal in an idealized variant of OCO (Zhang et al., 2022; 2024)).

Finally, the tuner requires a discount factor β. This crucially controls the strength of regularization, but also introduces a hyperparameter tuning problem. Following (Cutkosky, 2019), we aggregate tuners with different β (on a log-scaled grid) by simply summing up their outputs. This is justified by the *adaptivity* of the tuner itself: in OCO, if we add a parameter-free algorithm $\mathcal{A}_1$ to any other algorithm $\mathcal{A}_2$ that already works well, then $\mathcal{A}_1$ can automatically identify this and "tune down" its aggressiveness, such that $\mathcal{A}_1 + \mathcal{A}_2$ still performs as well as $\mathcal{A}_2$.

4 Experiment

We instantiate TRAC PPO with two different optimizers: ADAM with constant learning rate for baseline comparison, and TRAC for our proposed method (with exactly the same ADAM as the input BASE). We also test ADAM PPO with *concatenated ReLU activations* (CReLU; Shang et al., 2016), previously shown to mitigate loss of plasticity in deep RL (Abbas et al., 2023). More details and experiments can be found in Appendix A, H (including numerical results in Table 1).

OpenAI Procgen We evaluated TRAC on OpenAI Procgen, a suite of 16 procedurally generated games (Cobbe et al., 2019). Distribution shifts were introduced by sampling a new level every 2 million steps, treating each level as a distinct task. In StarPilot and Dodgeball, ADAM PPO and CReLU showed degrading performance with each new level (Figure 1). In contrast, TRAC PPO avoided this loss, achieving rapid reward increases. Overall, TRAC PPO demonstrated average improvements of 3,212.42% over ADAM PPO and 120.88% over CReLU (Table 1).

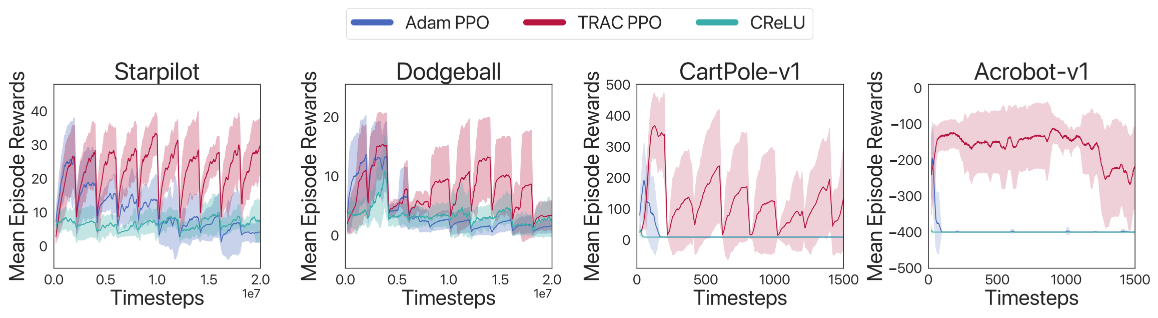

Figure 1: ADAM PPO and CReLU show a steady loss of plasticity in Procgen environments and fail to recover after the initial distribution shift in control environments. Conversely, TRAC PPO rapidly adapts to each shift.

Gym Control We use the CartPole-v1 and Acrobot-v1 environments from the Gym Classic Control suite. To introduce distribution shifts, every 200 steps we perturb each observation dimension with random noise within a range of ± 2, treating each perturbation phase as a distinct task. Here (Figure 1), we notice a peculiar behavior after introducing the first distribution shift in both ADAM PPO and CReLU: policy collapse. We describe this as an *extreme* form of loss of plasticity. Surprisingly, TRAC PPO remains resistant to these extreme distribution shifts. Across the three control environments, TRAC PPO shows an average normalized improvement of 204.18% over ADAM PPO and 1044.24% over CReLU (Table 1).

5 Conclusion

We introduced Trac, a parameter-free optimizer for lifelong RL using OCO principles. Trac dynamically refines regularization in a data-dependent manner, eliminating hyperparameter tuning. Trac's results lead to a compelling takeaway: empirical lifelong RL may exhibit more convex properties than previously appreciated, and might inherently benefit from parameter-free OCO approaches.

References

Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. *arXiv preprint arXiv:2303.07507*, 2023.

David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 10–19. PMLR, 10–15 Jul 2018a. URL https://proceedings.mlr.press/v80/abel18a.html.

David Abel, Yuu Jinnai, Sophie Yue Guo, George Konidaris, and Michael Littman. Policy and value transfer in lifelong reinforcement learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 20–29. PMLR, 10–15 Jul 2018b. URL https://proceedings.mlr.press/v80/abel18b.html.

Hongjoon Ahn, Jinu Hyeon, Youngmin Oh, Bosun Hwang, and Taesup Moon. Catastrophic negative transfer: An overlooked problem in continual reinforcement learning, 2024. URL https://openreview.net/forum?id=o7BwUyXz1f.

Kavosh Asadi, Rasool Fakoor, and Shoham Sabach. Resetting the optimizer in deep rl: An empirical study, 2023.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

Eseoghene Ben-Iwhiwhu, Saptarshi Nath, Praveen K. Pilly, Soheil Kolouri, and Andrea Soltoggio. Lifelong reinforcement learning with modulating masks, 2023.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *CoRR*, abs/1912.01588, 2019. URL http://arxiv.org/abs/1912.01588.

Ashok Cutkosky. Combining online learning guarantees. In *Conference on Learning Theory*, pp. 895–913. PMLR, 2019.

Ashok Cutkosky and Francesco Orabona. Black-box reductions for parameter-free online learning in banach spaces. In *Conference On Learning Theory*, pp. 1493–1529. PMLR, 2018.

Ashok Cutkosky, Aaron Defazio, and Harsh Mehta. Mechanic: A learning rate tuner. *Advances in Neural Information Processing Systems*, 36, 2023.

Shibhansh Dohare, Richard S Sutton, and A Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

Dylan J Foster, Satyen Kale, Mehryar Mohri, and Karthik Sridharan. Parameter-free online learning via model selection. *Advances in Neural Information Processing Systems*, 30, 2017.

Andrew Jacobsen and Ashok Cutkosky. Parameter-free mirror descent. In *Conference on Learning Theory*, pp. 4160–4211. PMLR, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Saurabh Kumar, Henrik Marklund, and Benjamin Van Roy. Maintaining plasticity via regenerative regularization. *arXiv preprint arXiv:2308.11958*, 2023.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks, 2023.

Jorge A. Mendez, Boyu Wang, and Eric Eaton. Lifelong policy gradient learning of factored policies for faster training without forgetting, 2020.

Golnaz Mesbahi, Olya Mastikhina, Parham Mohammad Panahi, Martha White, and Adam White. Tuning for the unknown: Revisiting evaluation strategies for lifelong rl, 2024.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 16828–16847. PMLR, 2022.

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and André Barreto. Deep reinforcement learning with plasticity injection, 2023.

Francesco Orabona. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*, 2023.

Francesco Orabona and Dávid Pál. Coin betting and parameter-free online learning. *Advances in Neural Information Processing Systems*, 29, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pp. 2217–2225. PMLR, 2016.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. *arXiv preprint arXiv:2302.12902*, 2023.

Zhiyu Zhang, Ashok Cutkosky, and Ioannis Paschalidis. Pde-based optimal strategy for unconstrained online learning. In *International Conference on Machine Learning*, pp. 26085–26115. PMLR, 2022.

Zhiyu Zhang, Heng Yang, Ashok Cutkosky, and Ioannis C Paschalidis. Improving adaptive online learning using refined discretization. In *International Conference on Algorithmic Learning Theory*, pp. 1208–1233. PMLR, 2024.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *International Conference on Machine Learning*, pp. 928–936, 2003.

Appendix

A Further Experiments and Numerical Results

Arcade Learning Environment (ALE) Atari The ALE Atari 2600 benchmark tests RL agents across diverse arcade games (Bellemare et al., 2012). We introduce distribution shifts by switching to a new game every 4 million timesteps, posing a greater challenge than OpenAI Procgen by requiring adaptation to changes in both state and reward.

We evaluated two settings with action spaces of 6 and 9. As shown in Figure 2, both Adam PPO and CReLU often failed in some games, while Trac PPO consistently increased rewards across different games. During the first 12 million steps (3 games) in Atari 6, Trac PPO achieved higher mean rewards and rapid increases. Overall, Trac PPO showed an average improvement of 329.73% over Adam PPO and 68.71% over CReLU (Table 1). In some instances, like the last 2 million steps of Atari 6, CReLU performed comparably to Trac PPO, aligning with findings that CReLU can prevent plasticity loss in continual Atari setups (Abbas et al., 2023).

LunarLander LunarLander-v2 is a physics-based control task from the Box2d gym control environment. As with the Acrobot and CartPole experiments (refer to Figure 1), we observe in Figure 3 that after the first distribution shift, both Adam PPO and CReLU fail to learn a stable policy. In contrast, Trac adapts rapidly even in these extreme distribution shifts, maintaining high performance throughout the task.

Chaser Chaser is another game from the Procgen suite. Similar to results in Dodgeball and StarPilot (refer to Figure 1), we observe in Figure 3 both Adam PPO and CReLU show degrading rewards due to loss of plasticity. However, Trac avoids this issue, demonstrating rapid reward increases and sustained performance across different levels.

Numerical results Our numerical results for the main experiments (Procgen, Atari, Gym Control) are summarized below in Table 1.

Table 1: Cumulative sum of mean episode reward for Trac PPO, Adam PPO, and CReLU on Procgen, Atari, and Gym Control environments. Rewards are scaled by 10^5; higher is better.

Environment	Adam PPO	CReLU	Trac PPO (Ours)
Starpilot	3.4	3.6	**12.5**
Dodgeball	1.9	2.3	**5.2**
Chaser	1.4	1.7	**2.2**
Fruitbot	0.1	1.0	**1.8**
CartPole	5.1	1.2	**39.6**
Acrobot	-14.3	-13.9	$\mathbf{-12.9}$
LunarLander	-21.7	-19.4	$\mathbf{-8.6}$
Atari 6	3.1	4.8	**10.5**
Atari 9	3.9	17.0	**20.2**

B Discussion

Related work Combating loss of plasticity has been studied extensively in lifelong RL. A typical challenge for existing solutions is the tuning of their hyperparameters, which requires prior knowledge on the nature of the distribution shift, e.g., (Asadi et al., 2023; Ben-Iwhiwhu et al., 2023; Nikishin et al., 2023; Sokar et al., 2023; Mesbahi et al., 2024). An architectural modification called CReLU is studied in (Abbas et al., 2023), but our experiments suggest that its benefit might be specific to the Atari setup. Besides, Abel et al. (2018a;b) presented a theoretical analysis of skill transfer in lifelong RL, based on value iteration.

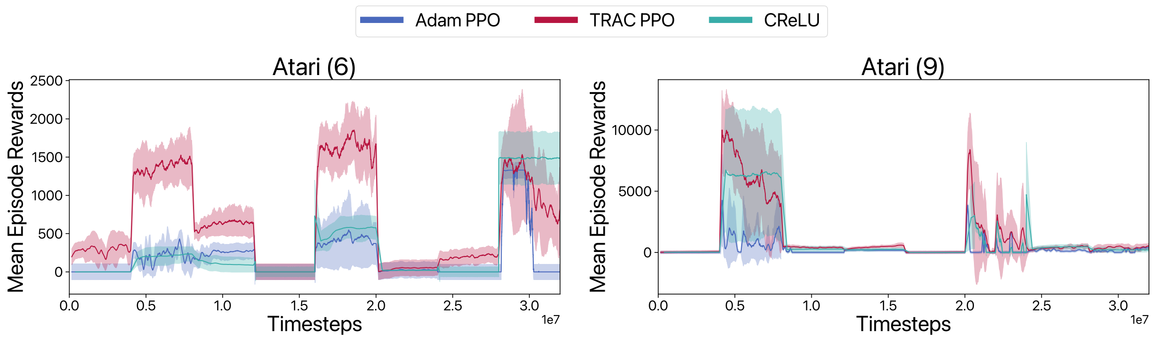

Figure 2: Reward in the lifelong Atari environments, across games with action spaces of 6 and 9. These plots demonstrate that TRAC PPO rapidly adapts to new tasks, in contrast to the ADAM PPO and CReLU which struggle to achieve high reward, indicating mild loss of plasticity.

Tuning L_2 regularization The success of TRAC suggests that adaptive regularization, anchored to θ_{ref}, effectively counters both mild and extreme loss of plasticity. This highlights the limitation of the L_2 regularization approach from (Kumar et al., 2023). It requires selecting a regularization strength parameter λ through cross-validation, which is incompatible with the one-shot nature of lifelong learning settings. However, even when we try the λ-grid suggested by (Kumar et al., 2023), there is no effective λ value within the grid for the lifelong RL environments we consider. All the values are too small. We conduct a hyperparameter search for λ, over various larger values $[0.2, 0.8, 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]$. We discover that each environment and task responds uniquely to these regularization strengths (Figure 4). This highlights the challenges of tuning λ in a lifelong learning context. In contrast, TRAC dynamically adapts to data online, consistently competing with well-tuned λ values in CartPole, Acrobot, and LunarLander (Figure 4).

On the choice of θ_{ref} In general, the reference point θ_{ref} should be good or "safe" for TRAC to perform effectively. One might presume that achieving this requires "warmstarting", or pre-training using the underlying BASE optimizer. While our experiments validate that such warmstarting is indeed beneficial (Appendix D), our main experiments show that even a random initialization of the policy's weight serves as a good enough θ_{ref}, even when tasks are similar (Figure 1).

This observation aligns with discussions by Lyle et al. (2023), Sokar et al. (2023), and Abbas et al. (2023), who suggested that persistent gradient steps away from a random initialization can deactivate ReLU activations, leading to activation collapse and loss of plasticity in neural networks. Our results

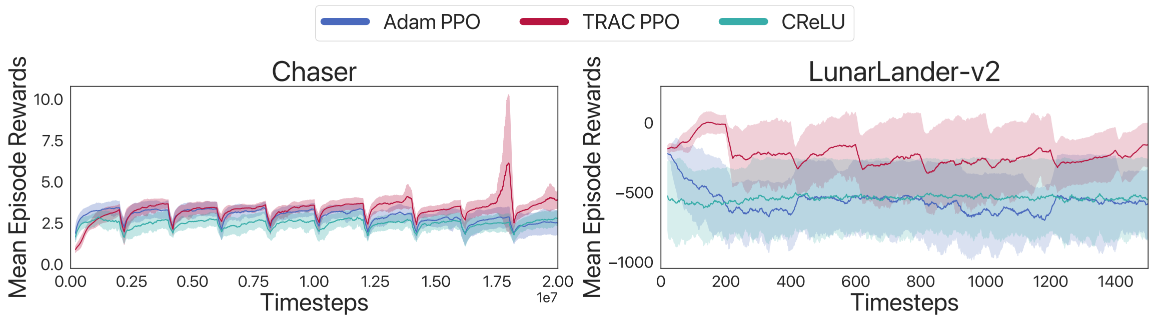

Figure 3: Reward plots for LunarLander-v2 and Chaser. In the Chaser environment, both ADAM PPO and CReLU show degrading rewards and fail to recover. In LunarLander, after the first distribution shift, ADAM PPO and CReLU are unable to learn a stable policy, while TRAC adapts rapidly and maintains high performance across tasks.

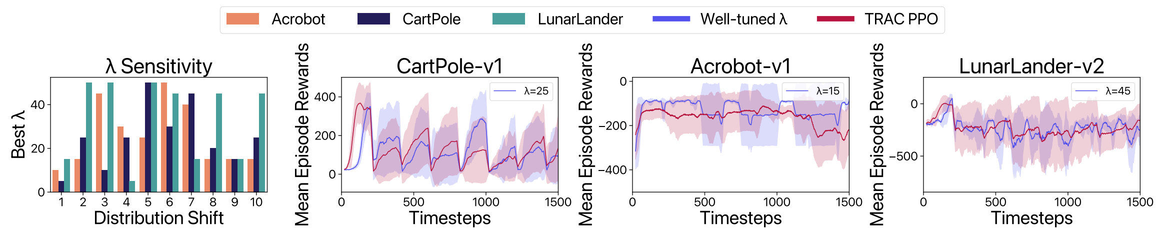

Figure 4: For each control environment and the initial ten tasks, we identified the optimal λ that maximizes reward for each task's specific distribution shift. We also determined the best overall λ for each environment: CartPole $\lambda = 25$, Acrobot $\lambda = 15$, and LunarLander $\lambda = 45$. The results show that different tasks and distribution shifts within each environment are sensitive to different λ values.

also support Kumar et al. (2023)'s argument that maintaining some weights close to their initial values not only prevents dead ReLU units but also allows quick adaptation to new distribution shifts.

Near convexity of lifelong RL Our results demonstrate the rapid adaptation of Trac, in lifelong RL problems with complicated function approximation. From the perspective of optimization, the latter requires tackling both nonconvexity and nonstationarity, which is typically regarded intractable in theory. Perhaps surprisingly, when approaching this complex problem using the theoretical insights from OCO, we observe compelling results. This suggests a certain "hidden convexity" in this problem.

C Trac Encourages Positive Transfer

To explore whether Trac encourages positive transfer, we introduce a privileged weight-reset baseline. This baseline is "privileged" in the sense that it knows when a distribution shift is introduced and resets the parameters to a random initialization at the start of each new task. We applied this baseline to three Gym control tasks: CartPole-v1, Acrobot-v1, and LunarLander-v2, and compared it to Trac PPO and Adam PPO, as shown in Figure 5.

We observe that the privileged weight-reset baseline exhibits spikes in reward at the beginning of each new task. Surprisingly, Trac maintains even higher rewards than the privileged weight-reset baseline, even at its peak learning phases. Additionally, Trac's reward does not decline to the reward seen at the start of new tasks with privileged weight-resetting (Trac does not have to "start over" with each task), suggesting that Trac successfully transfers skills positively between tasks.

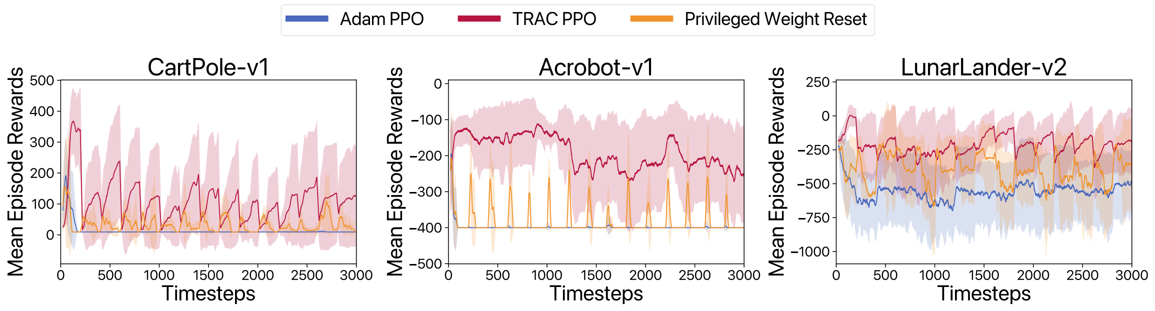

Figure 5: Reward comparison of Trac PPO, Adam PPO, and privileged weight-resetting on Cartpole-v1, Acrobot-v1, and LunarLander-v2. Trac PPO encourages positive transfer between tasks.

D Warmstarting

In our theoretical framework, we hypothesize that a robust parameter initialization, denoted as θ_{ref}, could enhance the performance of our models, suggesting that empirical implementations might benefit from initializing parameters using a base optimizer such as Adam prior to deploying Trac. Contrary to this assumption, our experimental results detailed in Section 4 reveal that warmstarting is not essential for Trac's success. Below, we examine the performance of Adam PPO and Trac PPO when warmstarted.

Both Trac PPO and Adam PPO were warmstarted using Adam for the initial 150,000 steps in all games for the Atari and Procgen environments, and for the first 30 steps in the control experiments. As seen in Figure 6, in games like Starpilot, Fruitbot, and Dodgeball, Trac PPO surpasses Adam PPO in the first "level" of the online setup, with its performance closely matching that of Adam PPO in Chaser. Importantly, Trac PPO continues to circumvent the loss of plasticity encountered by Adam PPO, even when both are warmstarted. This makes sense since all of the distributions share some foundational game dynamics; the initial learning phases likely explore these dynamics, so leveraging a good parameter initialization to regularize in this early region can be beneficial for Trac—we observe that forward transfer occurs somewhat in later level distribution shifts as the reward does not drop back to zero where it initially started from.

Our findings indicate that warmstarting does not confer a significant advantage in the Atari games. This makes sense because a parameter initialization that is good in one game setting is likely a random parameterization for another setting, which is equivalent to the setup without warmstarting where Trac regularizes towards a random parameter initialization. In the control experiments although warmstarted Trac PPO manages to avoid the extreme plasticity loss and policy collapse seen in warmstarted Adam PPO, it does not perform as well as non-warmstarted Trac PPO. This variability underscores that the efficacy of warmstarting is environment-specific and highlights the challenge in predicting when Adam PPO may achieve a parameter initialization that is advantageous for Trac PPO to regularize towards.

From an overall perspective, warmstarting Trac PPO in every setting still shows substantial improvement over Adam PPO (Table 2)

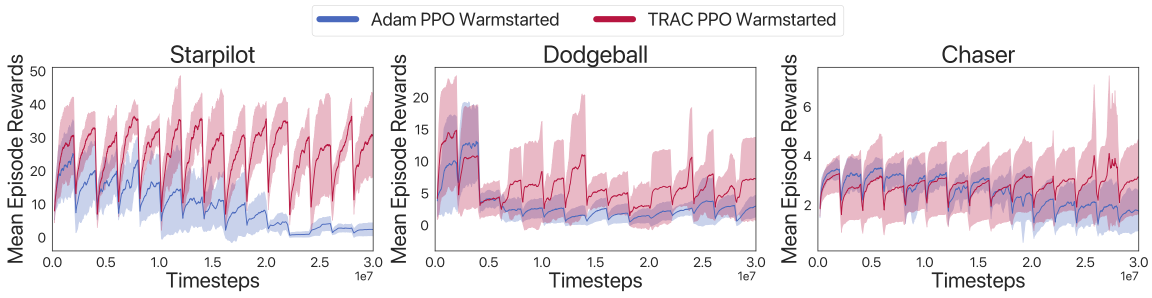

Figure 6: Comparison of reward in the lifelong Procgen environments for StarPilot, Dodgeball, Fruitbot, and Chaser with warmstarted Trac PPO and warmstarted Adam PPO. Inital performance of Trac PPO is improved with warmstarting and continues to avoid loss of plasticity.

E Gravity Based Distribution Shifts

One method to introduce distribution changes in reinforcement learning environments is by altering the dynamics Mendez et al. (2020), such as adjusting the gravity in the CartPole environment. In this set of experiments, we manipulate the gravity by a magnitude of ten, randomly adding noise for

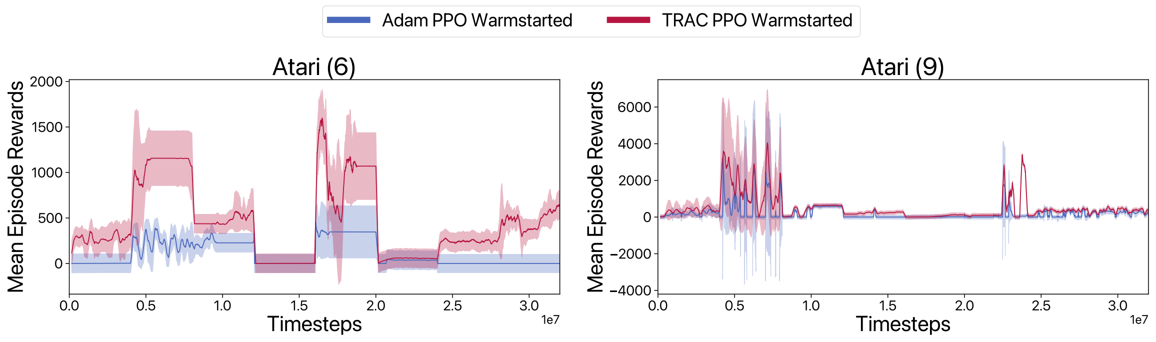

Figure 7: Comparison of reward in the lifelong Atari environments with warmstarted Trac PPO and warmstarted Adam PPO. No significant benefit is found by warmstarting Trac compared to not warmstarting it.

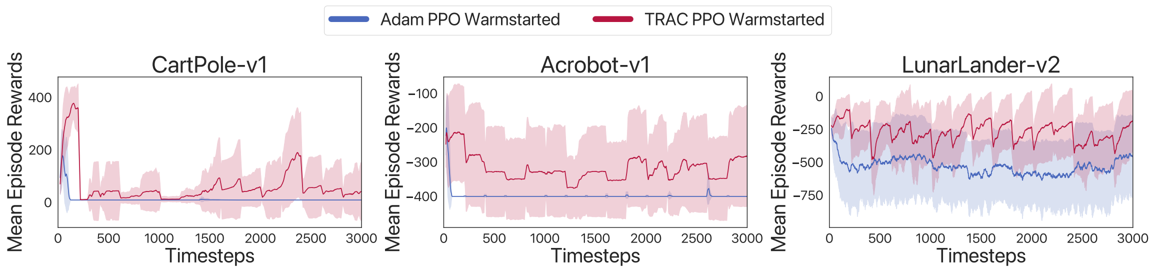

Figure 8: Comparison of reward in the lifelong Gym control environments for CartPole-v1, Acrobot-v1, and LunarLander-v2 with warmstarted Trac PPO and warmstarted Adam PPO.

Table 2: Cumulative sum of mean episode reward over all distributions for Adam PPO warmstarted and Trac PPO warmstarted on Procgen, Atari, and Gym Control environments. Rewards are scaled by 10^5; higher is better.

Environment	**Adam PPO**	**Trac PPO (Ours)**
Starpilot	3.0	**10.2**
Dodgeball	1.2	**2.5**
Chaser	1.3	**1.6**
Fruitbot	−0.4	**0.6**
CartPole	4.6	**22.8**
Acrobot	−142.9	**−114.5**
LunarLander	−190.7	**−97.3**
Atari6	16.7	**72.2**
Atari9	34.6	**80.6**

one distribution shift, and then inversely, dividing by ten and adding random noise for the next shift. This process continues throughout the experiment.

Our observations suggest that Adam PPO is robust to such dynamics-based distribution shifts, as shown in Figure 9. This indicates that while Adam PPO implicitly models the dynamics of the environment well—where changes in dynamics minimally impact performance—it struggles more with adapting to out-of-distribution observations such as seen in the main experiments (Figure **??**) and in the warmstarting experiments (Figure 8).

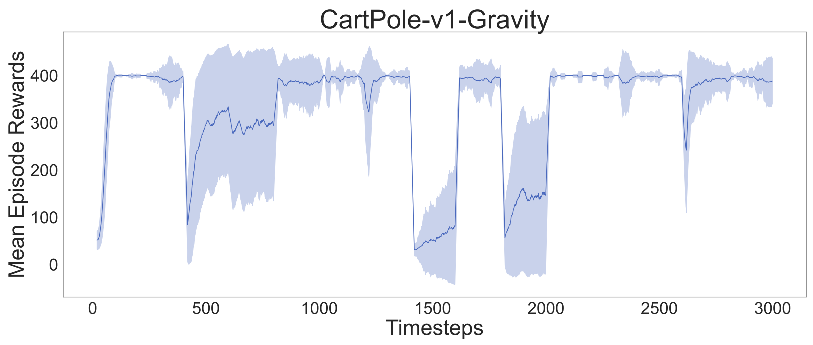

Figure 9: Mean Episode Reward for Adam PPO on CartPole-v1 with varying gravity. Adam PPO demonstrates robust policy recovery across most gravity-based distribution shifts.

F Scaling-Value Convergence

As discussed in the algorithm section (see Section 3), Trac operates as a meta-algorithm on top of a standard optimizer, denoted as Base. The crucial component of Trac involves the dynamic adjustment of the scaling parameter S_{t+1}, managed by the tuner algorithm (Algorithm 2). This parameter is data-dependent and typically ranges between $[0, 1]$. The weight update θ_{t+1} is consequently defined as a convex combination of the current optimizer's weight θ_t^{Base} and a predetermined reference point θ_{ref}.

This section presents the convergence behavior of the scaling parameter S_{t+1} across different environments, analyzed through the mean values over multiple seeds.

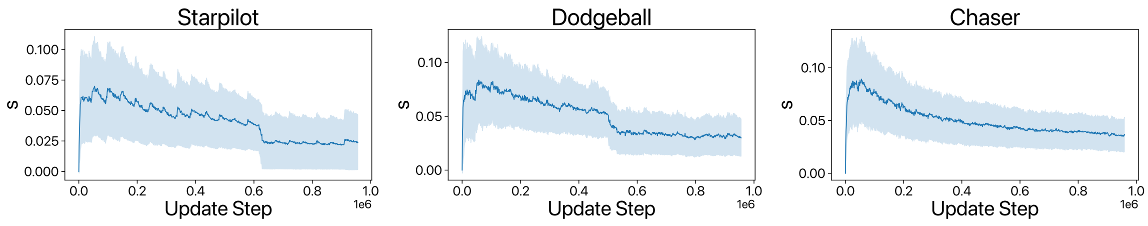

Figure 10: Convergence of the scaling parameter S_{t+1} in the Procgen environments.

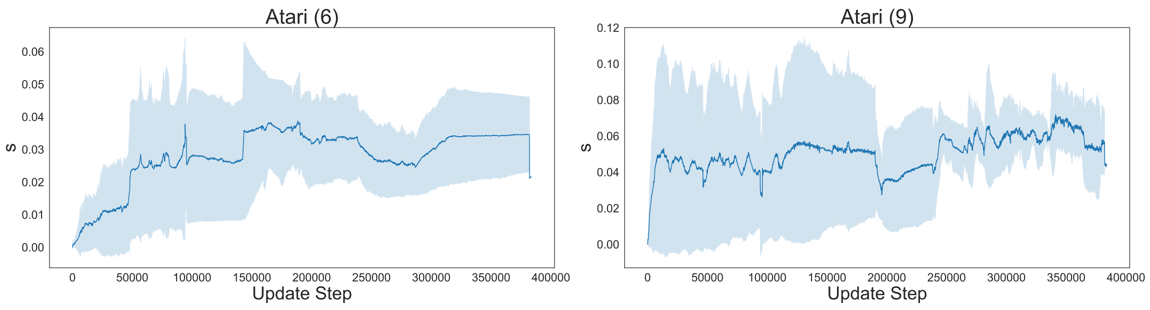

Figure 11: Evolution of the scaling parameter S_{t+1} in the Atari settings. Here we don't see a meaningful convergence ofS_{t+1}.

The convergence of the scaling parameter S_{t+1} observed across the ProcGen and Control environments, as depicted in Figures 10 and 12, reflects a good scaling value that effectively determines the strength

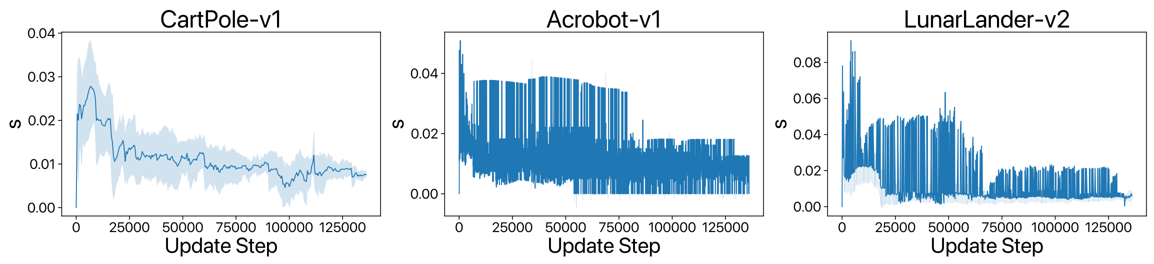

Figure 12: Convergence of the scaling parameter S_{t+1} in the Gym Control environments.

of regularization towards the initialization points, yielding robust empirical outcomes in lifelong RL settings. Interestingly, in ProcGen environments, this converged scaling value exhibits consistency across various games, typically hovering between 0.02 and 0.03, as shown in Figure 10. In contrast, in Control environments, the scaling values are notably lower, ranging between 0.005 and 0.01, as illustrated in Figure 12.

G Comparison to Mechanic

Our algorithm TRAC builds on a long line of works on parameter-free OCO (see Section 3). In particular, we are inspired by the MECHANIC algorithm. Compared to MECHANIC, TRAC improves the scale tuner there (which is based on the *coin-betting* framework; Orabona & Pál, 2016) by the erfi algorithm that enjoys a better OCO performance guarantee. We empirically compare TRAC and MECHANIC in the Starpilot game from the Procgen suite (Figure 13). The results indicate that while MECHANIC effectively mitigates plasticity loss and adapts quickly to new distribution shifts, it slightly underperforms in comparison to TRAC. This suggests potential for the effectiveness of the general “parameter-free” principle in lifelong RL.

H Experimental Setup

Procgen and Atari Vision backbone For both the Atari and Procgen experiments, the Impala architecture was used as the vision backbone. The Impala model had 3 Impala blocks, each containing a convolutional layer followed by 2 residual blocks. The output of this is flattened and connected to a fully connected layer. The impala model parameters are initialized using Xavier uniform initialization.

Policy and Value Networks Across all experiments—including Control, Atari, and Procgen—the policy and value functions are implemented using a multi-layer perceptron (MLP) architecture. This architecture processes the input features into action probabilities and state value estimates. The MLP comprises several fully connected layers activated by ReLU. The output from the final layer uses a softmax activation.

Trac TRAC, for all experiments, was implemented using the same experiment-specific baseline architectures and baseline optimizer. For the Procgen and Atari experiments, the base ADAM optimizer was configured as the same as baseline, with a learning rate of 0.001, and for Control experiments, a learning rate of 0.01 was used. Both learning rates were tested for all experiments and found to have negligible differences in performance outcomes. Other than the learning rate, we use the default ADAM parameters, including weight decay and betas, followed by the specifications outlined in the PyTorch Documentation.[2]

The setup for TRAC included β values for adaptive gradient adjustments: 0.9, 0.99, 0.999, 0.9999, 0.99999, and 0.999999. Both S_t and ε were initially set to (1×10^{-8}). Modifications were made

[2]https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

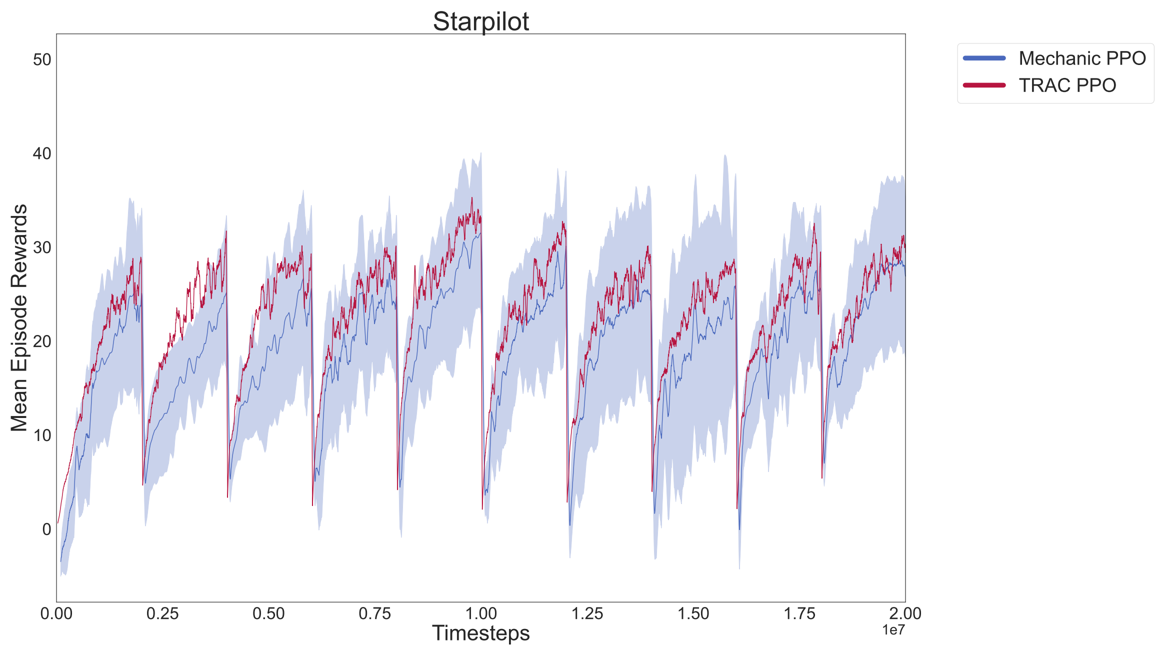

Figure 13: Comparison of reward in the lifelong StarPilot environment with both TRAC PPO and MECHANIC PPO. MECHANIC PPO performs similarly to TRAC PPO, although slightly underperforming TRAC PPO.

Table 3: PPO Parameters for Atari, Procgen, and Control Experiments

Parameter	**Atari**	**Procgen**	**Control**
Steps per update	2,000	1,000	800 (2 episodes with 400 steps)
Batch size	250	125	32
Epochs per update	3	3	5
Epsilon clip for PPO	0.2	0.2	0.2
Value coefficient	0.5	0.5	0.5
Entropy coefficient	0.01	0.01	0.01
Base Optimizer	ADAM (LR: 0.001)	ADAM (LR: 0.001)	ADAM (LR: 0.01)
Architecture	Impala + MLP	Impala + MLP	MLP

to a PyTorch error function library, which accepts complex inputs to accommodate the necessary computations for the imaginary error function. This library can be found at Torch Erf GitHub.[3]

Distribution Shifts In the Atari experiments, game environments were switched every 4 million steps. The sequence for games with an action space of 6 included “BasicMath”, “Qbert”, “SpaceInvaders”, “UpNDown”, “Galaxian”, “Bowling”, “Demonattack”, “NameThisGame”, while games with an action space of 9 included “LostLuggage”, “VideoPinball”, “BeamRider”, “Asterix”, “Enduro”, “CrazyClimber”, “MsPacman”, “Koolaid”.

For Procgen experiments, individual game levels were sampled using a seed value as the *start_level* parameter, which was incremented sequentially to generate new levels. Each new environment was introduced every 2 million steps.

[3]`https://github.com/redsnic/torch_erf`

In the Control experiments, each observation dimension was randomly perturbed by a value ranging from 0 to 2. This perturbation was constant for 200 timesteps, after which a new perturbation was applied, effectively switching the environmental conditions every 200 steps.

Statistical Significance Each game in the Procgen and Atari experiments was conducted with 8 seeds/runs, while the Control experiments utilized 25 seeds/runs. The exception was in the L_2 initialization experiments, which used 15 seeds/runs per regularization strength. In Figures 1, 2, 3, 4, 6, 7, 8, 9, the plotted lines represent the mean of all of the Mean Episode Rewards from the different seeds/runs, and the shaded error bands indicate the standard deviation of all of the Mean Episode Rewards from the different seeds/runs.

Compute Resources For the Procgen and Atari experiments, each was allocated a single A100 GPU, typically running for 3-4 days to complete. Control experiments were conducted using dual-core CPUs, generally concluding within a few hours. In both scenarios, an allocation of 8GB of RAM was sufficient to meet the computational demands.